CS2106 Lab 4

Sit nearer in front if you can :)

Ang Ray Yan

Lab 4 - High Level Overview

(Signalled via SIGUSR1 by MMU)

Exercise 1: Manage read-only memory

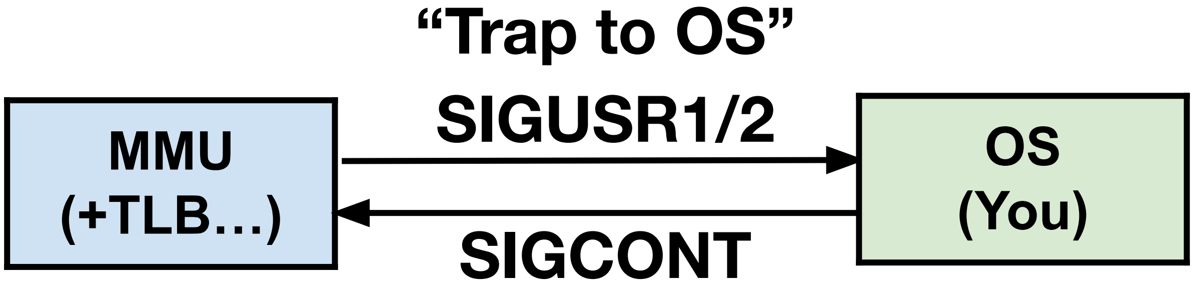
Exercise 2: Allow user process to write to memory

(Signalled via SIGUSR2 by MMU)

Exercise 3: Allow user process to alloc/de-alloc memory

Exercise 4: Commit-on-write

→ You are writing **os_run()**, signal SIGCONT to MMU...

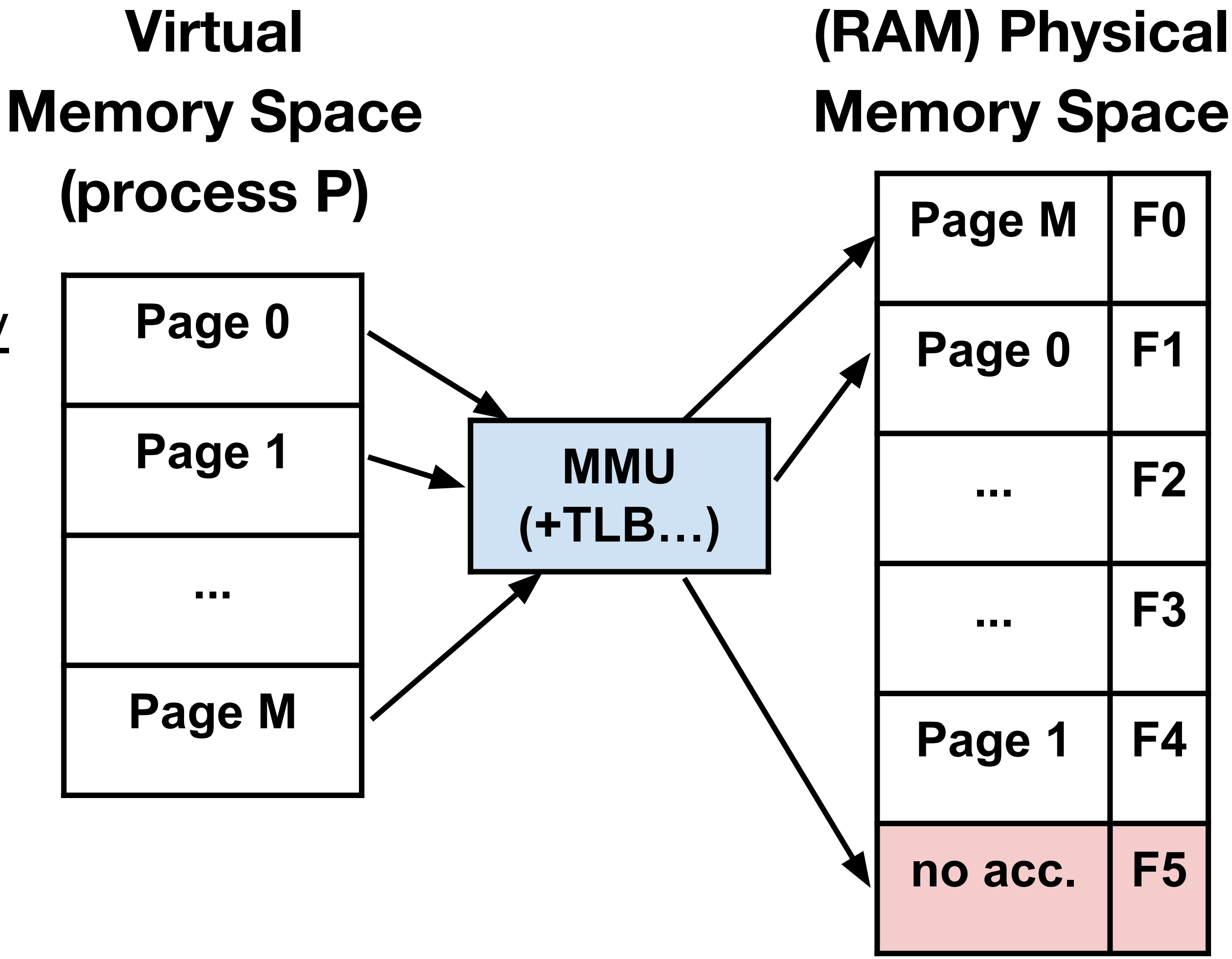


“Trap to OS”

Lab 4 - Memory management

Use virtual memory to protect physical memory :)

Focus here is on MMU / OS interaction...

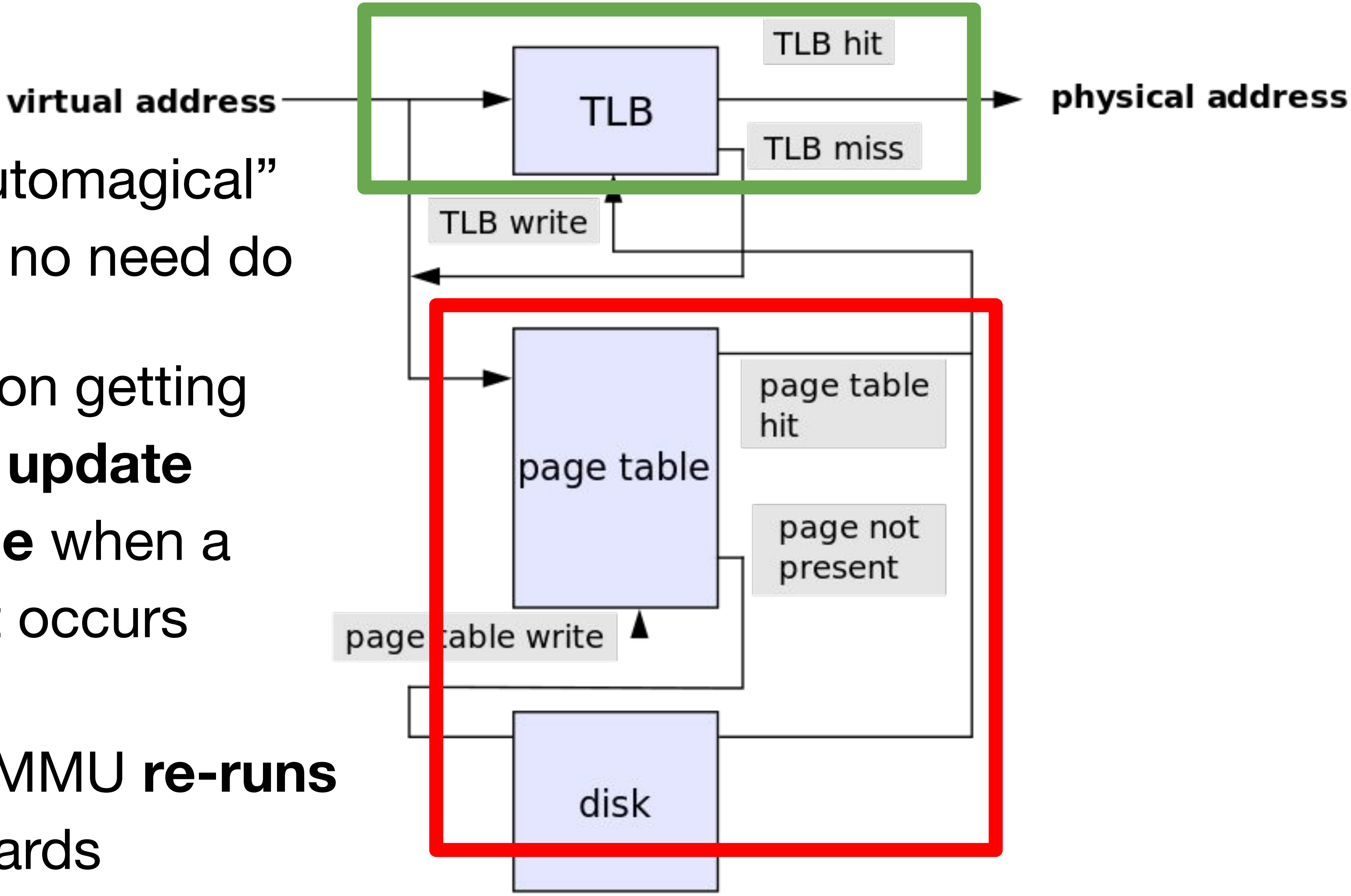


Lab 4 - MMU and Page Table

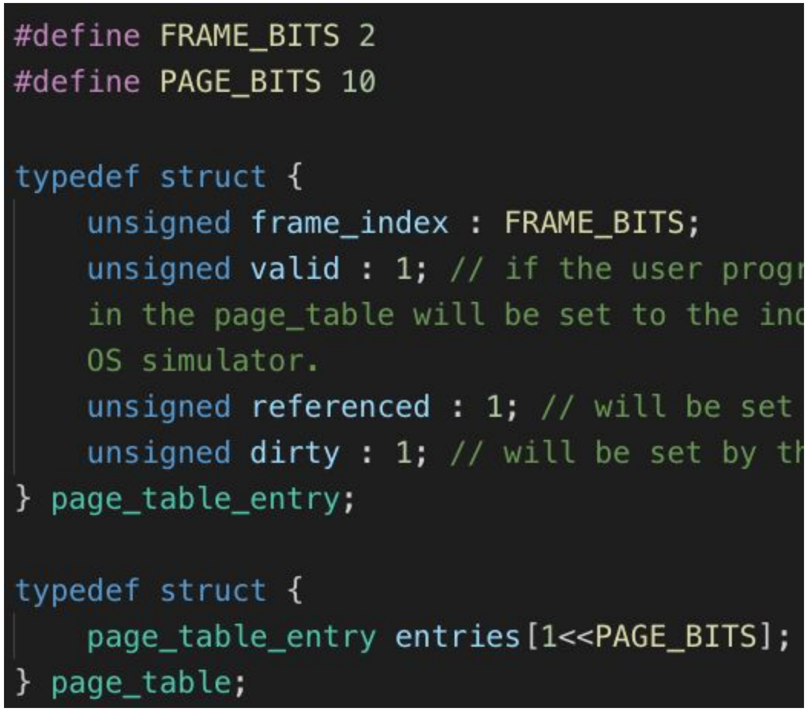
TLB is "automagical" for Lab 4, no need do

Focusing on getting the OS to **update page table** when a page fault occurs

- Note: MMU **re-runs** afterwards



Lab 4 - Page Table Entry (PTE)



```
#define FRAME_BITS 2
#define PAGE_BITS 10

typedef struct {
    unsigned frame_index : FRAME_BITS;
    unsigned valid : 1; // if the user progr
    in the page_table will be set to the ind
    OS simulator.
    unsigned referenced : 1; // will be set
    unsigned dirty : 1; // will be set by th
} page_table_entry;

typedef struct {
    page_table_entry entries[1<<PAGE_BITS];
} page_table;
```

Pg #	valid bit	ref. bit	dirty bit	frame index (e.g. 2 bits)
1	0	0	0	-
2	1	1	1	00
3	1	0	1	01
4	1	0	0	10
5	0	0	0	-

Frame Index: specifies frame to write to in ram

Valid: Is the page actually valid for user process to access

Referenced: MMU sets to 1 if page is accessed

Dirty: MMU sets to 1 if page is written to

Lab 4 - Ex1/2 (disk read + 2nd chance)

We want to read page #2

→ **valid bit = 1**

→ **Done** (no pg. fault)

→ MMU does not ask OS...

We want to read page #1

→ if frame 11 is **free…**

→ **disk-read**

→ valid bit = 1

→ frame index = 11

→ ref/dirty = 0

Pg #	valid bit	ref. bit	dirty bit	frame index (e.g. 2 bits)
1	0→1	0	0	11 (r)
2	1	1	1	00
3	1	0	1	01
4	1	0	0	10
5	0	0	0	-

Lab 4 - Ex1/2 (disk read + 2nd chance)

We want to read page #5

→ next_victim_frame=00

→ ref. bit=1, NVF++

  → decrement ref. bit

→ NVF=1, **evict page #3**

  → Dirty=1, **disk-write**

  → valid=0

→ For page #5, update

  → valid=1, ref/dirty=0

Pg #	valid bit	ref. bit	dirty bit	frame index (e.g. 2 bits)
1	1	0	0	11
2	1	1→ 0	1	00 ↓
3	0	0	1(w)	01
4	1	0	0	10
5	1	0	0	01

2nd chance: If ref-bit=1 due to MMU read, “2nd chance” will be given (i.e. ref→0) due to temporal locality instead during evict

Lab 4 - Ex3 - mmap/munmap

mmap(memory map)
→ Perform **Disk-create** for new page on the disk

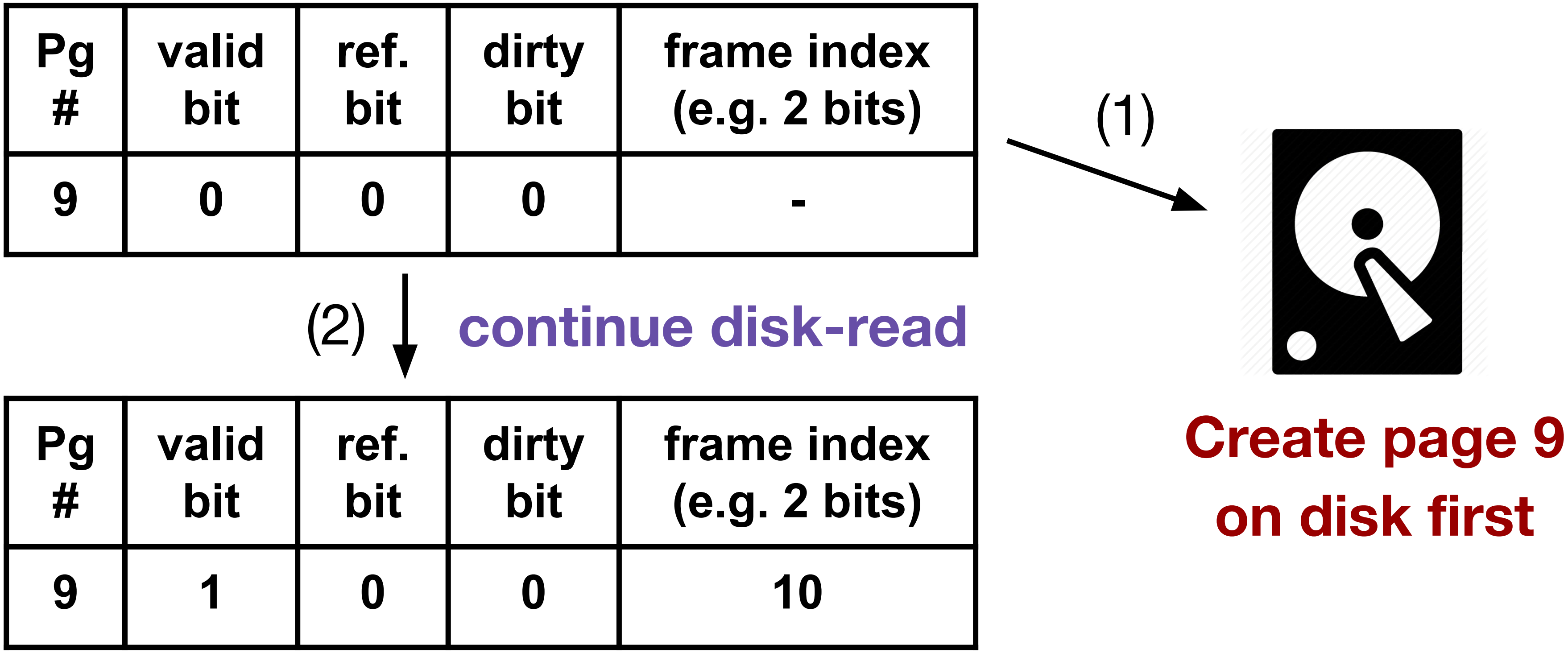
munmap (memory unmap)
→ perform **disk-delete** for specified page on the disk
→ Make sure page table is **consistent** after deleting...

Lab 4 - Ex4 - commit-on-write

Realize that **mmap/munmap** disk-create may be expensive...

→ e.g. mmap() then munmap() without any read/write

→ Solution: **disk-create just before 1st read/write (lazy)**



Pg #	valid bit	ref. bit	dirty bit	frame index (e.g. 2 bits)
9	0	0	0	-

(1)

Create page 9 on disk first

(2) **continue disk-read**

Pg #	valid bit	ref. bit	dirty bit	frame index (e.g. 2 bits)
9	1	0	0	10

Lab 4 - Conclusion (whose fault is it?)

Its PAGE FAULT because my program accessed memory in allocated space but is not in RAM currently

Its SEGMENTATION fault because my program actually kind of accessed memory outside of its allocated memory space

Its MY fault because I didn’t submit my CS2106 Lab 4 by Sat 2 Nov. 2pm

Lab 4 - Getting down to business

Lets start the demo :)

Order of demo: First raise hand and seen by me first serve

Things to get ready:

- Your compiled program for demo
- Your syntax-highlighted source code in text editor
- Explaining your program behaviours